VYSOKÁ ŠKOLA BÁŇSKÁ – TECHNICKÁ UNIVERZITA OSTRAVA

EKONOMICKÁ FAKULTA

KATEDRA APLIKOVANÉ INFORMATIKY

Testování webových aplikací

Testing of Web Applications

Student: Kateřina Pukowská

Vedoucí bakalářské práce: Ing. Jan Ministr, PhD.

Ostrava 2016

VŠB - Technická univerzita Ostrava
Ekonomická fakulta
Katedra aplikované informatiky

Zadání bakalářské práce

Student: **Kateřina Pukowská**

Studijní program: B6209 Systémové inženýrství a informatika

Studijní obor: 6209R017 Informatika v ekonomice

Téma: Testování webových aplikací
Testing of Web Applications

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Úvod
2. Teoretická východiska testování webových aplikací
3. Analýza stávajícího procesu testování
4. Návrh na inovaci procesu testování webových aplikací
5. Závěr

Seznam použité literatury
Seznam zkratek
Prohlášení o využití výsledků bakalářské práce
Seznam příloh
Přílohy

Seznam doporučené odborné literatury:

PATTON, Ron. *Software testing*. 2nd ed. Indianapolis: Sams Publishing, 2005. ISBN 0-672-32798-8.
ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
DOMES, Martin. *Tvorba WWW stránek pro úplné začátečníky*. Brno: Computer Press, 2008. ISBN 978-80-251-2160-3.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Ministr, Ph.D.**

Datum zadání: 20.11.2015
Datum odevzdání: 06.05.2016

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * OSTRAVA * Ekonomická fakulta 1

Ing. Petr Rozehnal, Ph.D.
vedoucí katedry

prof. Dr. Ing. Dana Dluhošová
děkanka fakulty

Čestné prohlášení

Prohlašuji, že jsem celou práci, včetně všech příloh, vypracovala samostatně.

V Ostravě dne 3. 5. 2016

Pukowská

.....

Kateřina Pukowská

Obsah

1 Úvod

Pro výběr daného tématu jsem se rozhodla na základě osobních zkušeností s prací softwarového testera a také jsem si chtěla prohloubit dosavadní nabyté zkušenosti v této oblasti.

Testování je jednou z důležitých součástí vývoje softwaru a mohli bychom ji popsat jako proces identifikování chyb. Na tuto část vývoje by se nemělo zapomínat, neboť díky ní firma může ušetřit nemalé finanční prostředky. Pokud se na problém přijde již během vývoje, je pro programátora jednodušší kód přepsat a také náklady na opravu jsou nižší. Navíc pokud se špatný produkt dostane mezi zákazníky, bude mít firma problémy nejen materiální, neboť bude muset zaplatit náklady na opravu, ale také bude poškozena pověst firmy.

V minulosti, kdy programy nebyly tak složité a pracovalo se v menších týmech, se nekladl příliš velký důraz na testování. Pokud se objevila nějaká chyba, dala se snadno opravit. Dnes jsou však projekty velké a odhalení omylů nemusí být vůbec jednoduché. Testování by se tedy mělo řádně naplánovat a stát se součástí vývoje SW, protože pokud se chyba objeví až po uvedení programu do běžného provozu, může mít katastrofální následky.

Je prakticky nemožné odhalit všechny chyby, které se v programu nacházejí, neboť vstupních dat může byt velmi mnoho, avšak měli bychom se snažit počet chyb minimalizovat, abychom tak maximálně uspokojili požadavky zákazníka. V současné době kvalita vyvíjených programů roste a zásluhu na tom má i uvědomění firem v tom, jak důležité testování může být.

Při vývoji softwaru by se také nemělo zapomínat na komunikaci. Program může být napsán sice správně, ale může být špatně pochopeno zadání. Proto je dobré, pokud se vývoje účastní i samotní uživatelé daného programu, otestují ho a vyjádří se k jeho fungování ještě před uvedením do ostrého provozu.

Cílem této práce je popsat teoretická východiska testování softwaru, provést analýzu stávajících procesů testování ve vybrané firmě a navrhnou jejich vylepšení.

Bakalářská práce je rozdělena na tři části. První část je teoretická a popisuje základní definice a pojmy, které se týkají testování softwaru. Jsou zde uvedeny například jednotlivé testovací techniky, jejich charakteristiky nebo role v rámci

testování. Druhá část se zabývá stavem testování softwaru, který slouží pro řízení vztahů se zákazníky, ve vybrané organizaci. V poslední části je popsán návrh na inovaci procesů tak, aby byly co nejefektivnější.

2 Teoretická východiska testování webových aplikací

2.1 Základní pojmy související s testováním webových aplikací

2.1.1 Testování

Hailpern (2002) popisuje testování jako jakoukoli aktivitu, která odhaluje chování programu, které je v rozporu se specifikací. Hlavním úkolem testera je provedení kódu na základě různých testovacích scénářů, tato práce však může zahrnovat i kontrolu návrhu, kontrolu samotného kódu apod.

Testování SW je důležité především proto, že na rozdíl od procesu výroby jiných produktů je vývoj softwaru velmi odlišný. Často se vytváří produkt, který je na míru určen konkrétnímu zákazníkovi a jeho potřebám. Z důvodu jedinečnosti těchto projektů dochází k chybám. Abychom si však mohli být jistí kvalitou produktu, je třeba do vývoje SW zahrnout také testování. (Roudenský, 2013)

2.1.2 Vytvoření specifikace

Před samotným vývojem softwaru musíme zjistit, jaké má zákazník na produkt požadavky. Ty nám však nic neřeknou o tom, jak má daný program vypadat. Z tohoto důvodu se vytváří specifikace, kde se uceleně sepíše vše, co programátoři a členové týmu potřebují vědět. Specifikace se mohou různě lišit v závislosti na tom, pro koho jsou vyvíjeny. Velice přesná specifikace se dělá především pro produkty, které budou sloužit státním úřadům, letecké nebo železniční dopravě, nebo například oboru lékařství. Tyto specifikace jsou pak pevně dané a kromě mimořádných událostí se již nesmí měnit. (Patton, 2002)

Na druhou stranu jsou i případy, kdy se specifikace nepíše. Tyto týmy se snaží být velmi flexibilní, riskují však, že dojde k nepředvídatelným problémům. Výsledná podoba produktu je tedy známá až po uvedení na trh. (Patton, 2002)

Při psaní specifikací se často dělají chyby. Může se stát, že není dostatečně podrobná nebo se neustále mění, což vede ke vzniku omylů. Další příčinou může být to, že je nesprávně pochopena. Ve specifikaci bychom se měli vyvarovat větám, které jsou nejednoznačné. Jako příklad můžeme uvést větu „Při souboji s koncovým nepřítelem bude po zničení hráč přesunut do lokace XYZ“. Z této věty není patrné, zda do lokace XYZ bude přesunut vítěz, či poražený. (Patton, 2002, Roudenský, 2013)

2.1.3 Testovací případ (test case)

„Testovací případ (test case) je podle IEEE 1012:2004 definován jako sada vstupů, podmínek pro spuštění a očekávaných výsledků, vyvinutá pro konkrétní účel, jako je provedení specifické cesty v programu nebo ověření souladu s konkrétním požadavkem.„

V mnoha organizacích dochází k situaci, kdy dokumentace není požadována, proto se opomíjí. Firma se však tímto úkonem vzdává mnoha výhod, jako je například snadné provádění regresních testů, častější odhalení chyb ve specifikaci, možnost kontroly testování a návrhů testů apod. (Roudenský, 2013)

Při psaní testovacích případů bychom měli dbát na to, že test case ověřuje pouze jednu situaci. Neměly by se zde složitě kombinovat podmínky, neboť může dojít k nepochopení a chybě při testování. Testovací případy by měly být tak jednoduché a přesné, aby je dokázal provádět kdokoli bez znalosti systému. Při opakovaném testování jednoho testovacího případu bychom měli vždy dojít ke stejnému výsledku. Pokud dojde k nějaké změně v systému, je třeba testovací případy aktualizovat, případně smazat, pokud již nejsou potřebné. (Roudenský, 2013)

2.1.4 HTML

„HTML je zkratkou pro Hypertext Markup Language, česky hypertextový značkovací jazyk. Je to jazyk, jenž se skládá ze značek a jehož smyslem je umožnit zobrazení textového a obrazového obsahu stránek a propojit jednotlivé stránky mezi sebou pomocí tzv. odkazů.“ (Domes, 2008, s. 18)

Jazyk HTML slouží k tomu, aby WWW stránky měly smysl. Díky něho mají stránky logickou strukturu. Víme například, co je nadpis, která část textu patří do jednoho odstavce, ale už nevíme, jaká je barva písma nebo jeho velikost. (Domes, 2008)

2.1.5 CSS

„CSS je zkratkou Cascading Style Sheets, česky šablony kaskádových stylů (zkráceně kaskádové styly). Byl vyvinut později než HTML s cílem vymýtit z jazyka HTML jakékoli prezentační, tedy vzhledové prvky. CSS samotný poskytuje vše podstatné pro určení vhledu kterékoli části WWW stránky.“ Jazyk CSS je jazykem stylovacím. (Domes, 2008, s. 19)

2.1.6 Verifikace a validace

V souvislosti s testování je nutné si vymezit pojmy verifikace a validace. Tyto koncepty jsou si velmi blízké, a proto se často zaměňují. Verifikace a validace slouží k tomu, abychom zjistili, zda software odpovídá specifikaci a dělá to, co od něj uživatel očekává.

IEEE Standard Glossary of Software Engineering Terminology definuje verifikaci a validaci následovně:

- „verifikace je proces určení toho, zda produkt v dané fázi vývojového cyklu splňuje požadavky stanovené během předchozí fáze,
- validace je proces hodnocení softwaru na konci jeho vývoje z hlediska zajištění požadavků na software.“ (IEEE 610.12, 1990, s. 80, 81)

Neformálně definuje Barry W. Boehm verifikaci a validaci pomocí následujících otázek:

- verifikace: Vytvářím produkt správně?
- validace: Vytvářím správný produkt?

I když některé především starší definice uvádějí, že validace probíhá až na konci vývoje softwaru, ne všichni autoři s tímto tvrzením souhlasí. Příkladem může být V-model, kde verifikace a validace probíhá již od začátku vývoje. (Roudenský, 2013)

2.2 Selhání softwaru

Na vývoji softwaru se většinou podílí mnoho lidí a kdokoli z nich může ve své práci udělat chybu. Díky tomu dochází k situacím, kdy program nefunguje tak, jak by měl.

2.2.1 Chyba softwaru

Selhání můžeme nazvat několika výrazy, jako jsou například vada, závada, problém, omyl, událost, anomálie, odchylka, selhání, nekonzistence nebo chyba. Existuje mnoho názvů, avšak každý z nich může mít trochu jiný význam.

Závada, selhání a vada se používají při situacích, které jsou opravdu nebezpečné, kdežto anomálie, událost nebo odchylky nejsou až tak závažné a mají vyjádřit spíše neúmyslnou chybu nebo nedomyšlenou činnost. V bakalářské práci se však budou vyskytovat spíše obecnější pojmy, jako jsou problém, omyl, chyba nebo bug.

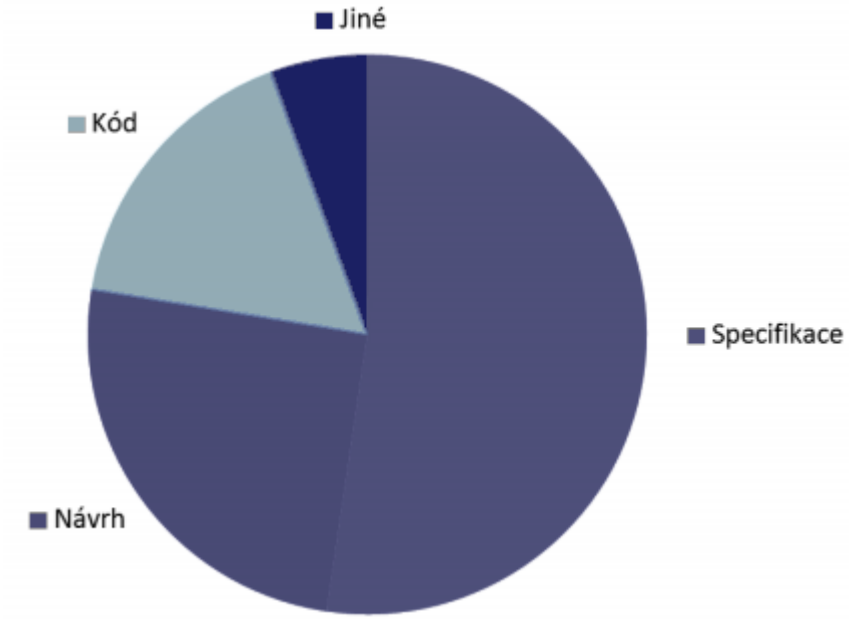
Patton (2002, s. 14) ve své knize uvádí, že o chybě hovoříme, pokud je splnění alespoň jedna z následujících podmínek:

- „software nedělá něco, co by podle specifikace produktu dělat měl,
- software dělá něco, co by podle údajů specifikace produktu dělat neměl,
- software dělá něco, o čem se produktová specifikace nezmiňuje,
- software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat,
- software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo – podle názoru testera softwaru – jej koncový uživatel nebude považovat za správný.“

Tato definice je napsána velmi obecně, aby bylo zajištěno to, že budou všechny omyly nalezeny. (Patton, 2002)

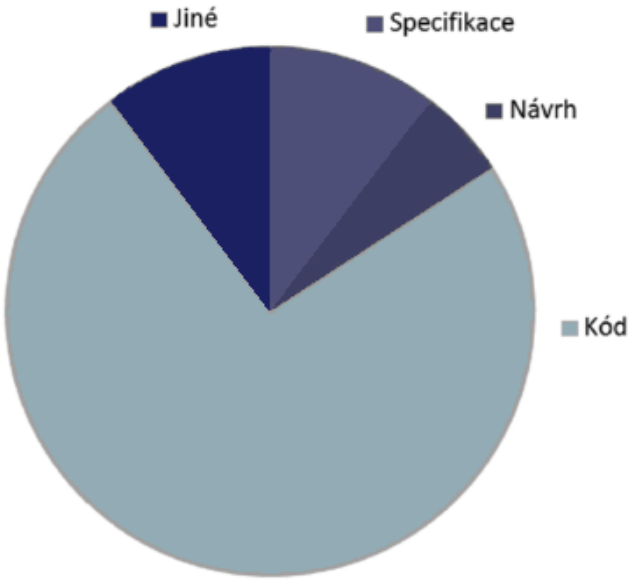
2.2.2 Důvody vzniku chyb

Patton (2001) tvrdí, že hlavním důvodem vzniku chyb je špatná specifikace. Tyto specifikace buď nejsou vůbec napsány, nebo jsou nesrozumitelné, málo podrobné nebo se mění tak často, že mohou být pro vývojářský tým matoucí. Jako druhý největší zdroj chyb uvádí návrh softwaru.



Obr. 2.1 – zdroj: Patton, 2002

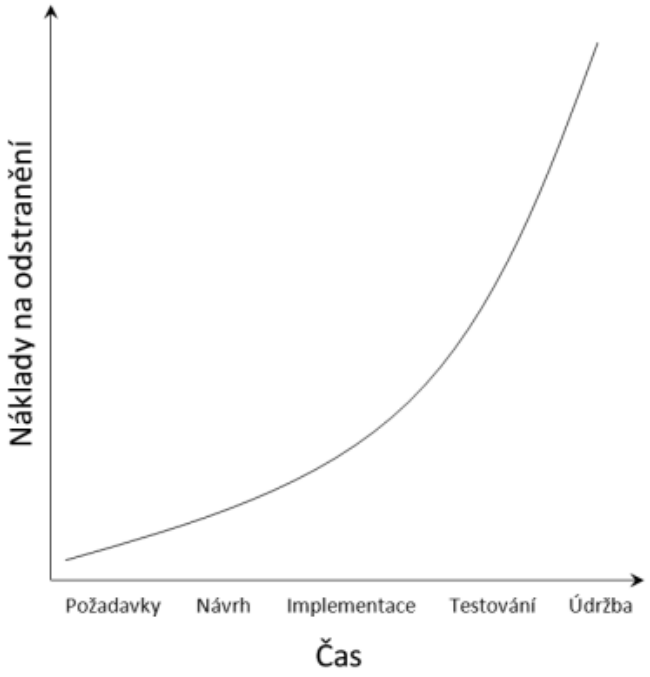
Novější zdroje však ukazují, že 70-80% chyb vzniká ve zdrojovém kódu, což je nejspíš následek toho, že firmy začaly používat vhodné metodiky, dodržovat normy a využívat lepší nástroje pro analýzu a návrh systému. (Roudenský, 2013)



Obr. 2.2 – zdroj: Roudenský, 2013

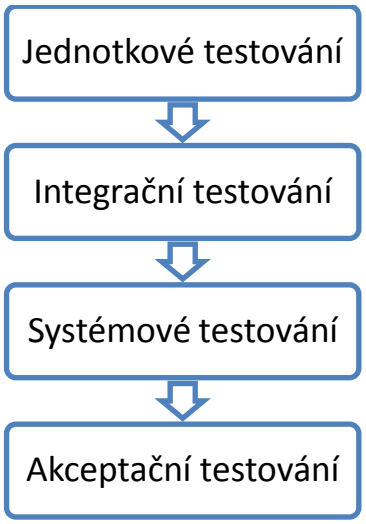
2.2.3 Náklady na opravu chyb

Software můžeme testovat v různých fázích jeho vývoje. Obecně platí, že čím dříve chybu najdeme, tím méně prostředků musíme vynaložit na jejich opravu. Pokud chybu objevíme už při psaní specifikace, nestojí nás její oprava skoro nic. Pokud však chybu objeví až koncový uživatel, je oprava velmi nákladná. (Patton, 2002)



Obr. 2.3 – zdroj: Roudenský, 2013

2.3 Techniky testování podle úrovně vývoje



Obr. 2.4 - techniky testování podle úrovně vývoje

2.3.1 Unit testing – jednotkové testování

Testování jednotek nebo také testování modulů představuje kontrolování malé, izolované části kódu, kterou nejčastěji provádí sám programátor. Unit testy musí být navrženy tak, aby se daly často opakovat a aby byly rychlé. Může se totiž stát, že takový test spustíme i tisíckrát a více. Jelikož tyto testy bývají jednoduché a snadné na údržbu, jsou i náklady na jejich provoz nízké. Příkladem jednotek mohou být funkce, procedury, metody nebo třídy. Jejich testováním tedy zjistíme chyby, které se nacházejí v jednotlivých částech kódu, nikoli chyby vzniklé při integraci do systému jako celku. (McWherter, 2010, Roudenský, 2013)

2.3.2 Integration testing – integrační testování

Na první pohled se muže zdát, že integrační testy jsou podobné jako testy jednotkové, ale není tomu tak. Hlavním úkolem integračních testů je zjistit, zda jednotlivé komponenty mezi sebou dobře spolupracují. To znamená, že jednotlivé moduly se spojí do většího celku a ten se pak otestuje. Integrační testy jsou mnohem náchylnější na chyby než unit testy a proto by se měly oddělovat. Jestliže selže unit test, je jasné, že došlo k chybě programu, pokud však selže integrační test, musíme zkontrolovat, jaká byla příčina. Může se totiž stát, že náš program je v pořádku, pouze jsme se zrovna nemohli připojit do databáze apod. Dalším důvodem, proč se od sebe tyto testy liší, je rychlost. Integrační testy jsou pomalejší, zato se nespouštějí tak často. (Patton, 2002, McWherter, 2010)

2.3.3 System testing – testování systému

Po dokončení integračních testů máme dostatečně stabilní systém, na němž se v jednom velkém procesu provede systémový test, tedy test celého systému. Tyto testy se provádějí v pozdějších fázích vývoje softwaru. Tato část je klíčová, neboť díky ní můžeme naposledy odhalit chyby ještě před tím, než se produkt dostane ke koncovému uživateli. Systémové testování zahrnuje například bezpečnostní testy, testy robustnosti, kterými testujeme, jak se systém dokáže přizpůsobit chybným vstupním datům nebo neočekávaným situacím, testy spolehlivosti nebo testy výkonnosti. Cílem systémové testování není pouze nalézt chyby v systému, ale také ověřit si jeho funkčnost. (Patton, 2002, Roudenský, 2013)

2.3.4 Acceptance testing – akceptační testování

Při vývoji softwaru hraje velmi významnou roli zákazník. i když si myslíme, že jsme napsali program správně, je důležité, aby si to myslel i uživatel. Proto bychom s ním měli být v kontaktu během celého vývoje. Pokud dáme zákazníkovi až hotový program, může se stát, že mu nebude vyhovovat a následná oprava takových chyb může dosti prodražit celý projekt.

Akceptační testování (UAT – User Acceptance Testing) je zaměřené na zákazníka a jedná se o nejdůležitější validační aktivitu. Zákazníkem může být jakýkoli zainteresovaný subjekt, ale většinou se jedná o koncového zákazníka, který bude program používat. Během akceptační testů není naším úkolem zjistit, zda systém funguje přesně tak, jak má, ale zda splňuje požadavky klienta. Ty jsou předem dohodnuty a jejich nesplnění, může vést k odmítnutí produktu. Standardně se vytváří plán akceptačního testování, na jehož základě se pak provádějí testy softwaru, ale také testy ostatních dodaných součástí jako jsou manuály apod. Během testů se uživatelé zaměřují především aktivity z reálného provozu. (Roudenský, 2013, McWherter, 2010)

2.4 Ostatní techniky testování

2.4.1 Ad hoc a explorativní testování

Ačkoli je ad hoc a explorativní testování velmi podobné a mnoho lidí je považuje za totožné, můžeme je přesto od sebe navzájem rozlišit.

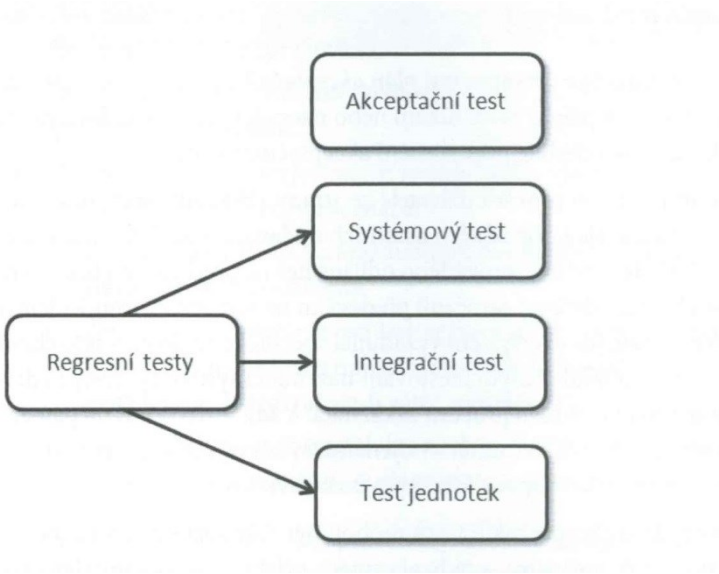
Ad hoc testování probíhá bez jakékoli přípravy. Tester nesystematicky prochází program a snaží se najít co nejvíce chyb. Nevytváří se žádná dokumentace a je pokryta jen malá část aplikace. Ačkoli se jako ad hoc označuje neefektivní testování, jeho úspěšnost není zanedbatelná a může tak vhodně doplnit ostatní testovací techniky.

Explorativní přístup je na rozdíl od ad hoc systematický. Začíná se zmapováním všech podstatných informací o produktu, identifikují se problémové části programu. Rozhodne se, kterým oblastem se budeme věnovat podrobně, kterým méně nebo jestli se některá z nich vynechá. Mělo by se také rozhodnout, zda nás zajímá funkčnost, bezpečnost nebo například spolehlivost. Poté, co se tyto věci vyřeší, pustí se tester na základě svých znalostí a zkušeností a také použité technologie do sepisování a provádění testovacích případů. Obě činnosti se provádí ihned po sobě. Tester se tak

může rozhodnout, zda se danou oblastí bude zabývat hlouběji nebo bude pokračovat dál a hledat chybu v jiné části programu. (Roudenský, 2013)

2.4.2 Regresní testování

Během vývoje softwaru často dochází k situaci, kdy se musí přidat nebo naopak odstranit některé funkcionality případně kód. Tyto změny sebou nesou jisté riziko, které bychom měli brát v úvahu. Pro tyto případy slouží regresní testy, které se snaží zjistit, zda v již otestovaném systému nedošlo k zavlečení nové chyby a systém tak zůstává stále funkční. Regresním testování se provádí na jednotkové, integrační i systémové úrovni a lze při nich využít již napsaných testovacích případů.



Obr. 2.5 – zdroj: Roudenský, 2013

Jelikož je regresní testování nákladná činnost, neprovádí se většinou při každé změně. Vyberou se pouze ty testy, které mohou pokrýt co největší oblast, ve které mohlo dojít k chybě. Mimo to se vytváří i balík regresních testů, v němž jsou testy, které se provádějí pravidelně. o tom, které testy budou součástí tohoto balíčku, rozhodne daná organizace. (Roudenský, 2013)

2.4.3 Testování černé, šedé a bílé skříňky

Testování černé a bíle skříňky vychází z anglických slov „black box“ a „white box“ (někdy také „glass box“).

Podstatou testování černé skříňky je to, že neznáme procesy, které se provádějí uvnitř programu po jeho spuštění. Můžeme tedy zadat vstupní data a očekávat určité

výstupy. Pokud se jedná o matematický výpočet, může si softwarový tester výsledky ověřit například na kalkulačce. Během testování tedy můžeme program využít pouze tak, jak to může udělat běžný koncový uživatel.

Jako protiklad můžeme uvést testování bílé skříňky (někdy označováno jako testování průhledné skříňky). Při tomto testování tester může nahlédnout do kódu a zjistit jeho slabá místa, neboť rozumí procesům, které se při běhu programu odehrávají. Znalost kódu a jeho spojitostí nemusí být vždy výhodou. Může se stát, že tester podvědomě zadá pouze taková vstupní data, která budou vyhovovat a nezjistí se chyba programu. Musíme být obezřetní a snažit se testovat objektivně. (Ash, 2003, Patton, 2002)

Směsi testování černé a bíle skříňky je testování šedé skříňky. Software testujeme, jako černou skříňku, ale částečně taky do něho nahlížíme. Příkladem může být nahlédnutí do zdrojového kódu, který je napsat v jazyce HTML. Jelikož tento jazyk je jazykem značkovacím a ne programovacím, jedná se o testování šedé skříňky. (Patton, 2002)

2.4.4 Manuální a automatizované testování

Manuální testování je prováděno člověkem, automatizované softwarem. Většina modelů vývoje softwaru obsahuje cyklus kódování – testování – oprava a ten se může i několikrát opakovat. Jednou z možností, jak si tuto práci ulehčit je napsat automatický test. Před tím, než se však pro tento test rozhodneme, měli bychom se zamyslet, zda daný testovací případ je vůbec vhodný pro zautomatizování. Důležité jsou především tyto faktory:

- rychlost,
- efektivita,
- správnost a přesnost,
- neúnavnost.

Tyto faktory jsou důležité především proto, že mohou ovlivnit náklady, které na testování vynaložíme. Například musíme zvážit, kolikrát chceme automatický test spustit a jak dlouho nám bude trvat jeho napsání. Může se stát, že provedení manuálního testu zabere kratší čas. Pokud tomu tak ale není, zbyde vám více času na jinou práci a zaměstnanec tak může třeba více promyslet testovací případ nebo pracovat

na něčem jiném. Jelikož jsme jen lidé, můžeme se unavit a začít dělat chyby. To se počítači stát nemůže, proto pokud je test napsán správně, můžeme se spolehnout, že práci vykoná tak, jak má.

Měli bychom však brát v potaz, že automatický test nikdy nemůže plně nahradit softwarového testera. Může mu však ulehčit jeho práci. (Patton, 2002)

2.4.5 Statické a dynamické testování

Statické testování je takové testování, kdy program neběží a my ho prohlížíme a kontrolujeme. Jako příklad statického testování můžeme uvést testování specifikací nebo revize zdrojového kódu. Statická analýza je typickou verifikační aktivitou a může se provádět již během ranné fáze vývoje softwaru.

Dynamické testování je oproti statickému mnohem více rozšířené. Program se spustí a my poté s ním pracujeme. Zadávají se vstupní data a kontrolují se výstupy. Zjišťujeme tak, jak se software chová a jaké jsou jeho vlastnosti. (Roudenský, 2013)

2.4.6 Testování splněním a selháním

Během testování splněním (anglicky test-to-pass) se snažíme zjistit, zda program vykazuje alespoň minimální funkčnost. Zadávají se taková vstupní data, která by se za ideálních podmínek měla vkládat a nesnažíme se hnát program za hranice jeho možností. Aplikujeme tedy ty nejjednodušší testové případy.

Teprve poté, co se přesvědčíme, že za normálních podmínek software dělá, co má, přechází se k testování selháním neboli vynucením chyb. Při tomto testování se snažíme najít slabá místa a vyvolat chybu programu, například zadáním hraničních podmínek. (Patton, 2002)

2.5 Lidský faktor

V současnosti se od testerů očekává, že budou mít i základní znalosti programování. Může se stát, že tyto znalosti tester ke své práci nepotřebuje a nevyužije je. Od profesionálů se však tato znalost očekává. Může ji využít při psaní automatických nebo zátěžových testů. Další výhodou je, že tento tester může snáz odhadnout slabá místa systému a to, kde programátor mohl udělat chybu. Na rozdíl od programátora tedy nemusí znát programovací jazyk zcela dopodrobna.

Při testování se také často můžeme setkat s využíváním relačních databází, z tohoto důvodu je dobré, aby tester znal jazyk SQL, případně jeho procedurální rozšíření. V neposlední řadě je také důležitá znalost podpůrných nástrojů, díky nimž je tester schopen samostatně pracovat na zadaných úkolech. (Roudenský, 2013)

2.6 Role v rámci testování

Pouze vzácně je testování přiděleno jednomu pracovníkovi. Obvykle se tým skládá z několika členů, kteří mají přidělené své role. u středních a větších projektů se objevují většinou tyto role:

- **tester** – provádí testy na základě připravených scénářů, zaznamenává výsledky a vytváří hlášení o nalezených chybách. Po nápravě defektů zajišťuje opakované otestování původního scénáře. Dále se také může starat o správu testovacího prostředí – funkčnost, konzistence dat apod.,
- **analytik testování (Test analyst nebo Test engineer)** – vytváří testovací případy a testovací scénáře. Musí tedy zanalyzovat požadavky a vytyčit případy, které je třeba otestovat. Ty sepíše a přiřadí jim prioritu. Dále může vytvářet regresní testy, starat se o jejich údržbu nebo vytvářet hlášení o vytvořených testech,
- **tester – automatizované testování (Test automator)** – spolupracuje s analytiky testování a automatizuje vhodné případy nebo celé scénáře, vytváří dokumentaci, zaznamenává výsledky nebo spravuje testovací prostředí. Obdobně to funguje u testerů, kteří se zaměřují na zátěžové testy,
- **vedoucí testování (Test lead)** – zodpovídá za testovací tým, vytváří plán testování, stará se o systém pro hlášení chyb, zadává úkoly a dohlíží na jejich plnění, vytváří hlášení o celkovém stavu testování. Může se účastnit psaní a provádění testů. Vedoucí by měl mít dostatečné technické znalosti, praxi v testování a zkušenosti s řízením lidí,
- **manažer testování (Test manager)** – je na vrcholu pomyslné pyramidy a je zodpovědný za vše, co souvisí s testováním. Volí přístup k testování, stanovuje cíle a kritéria měření jejich naplnění, vyjednává s managementem, rozhoduje o sestavení týmu atd. V praxi se může stát, že je tato role sloučena s rolí vedoucího testování, především u menších projektů. (Patton, 2002)

3 Analýza stávajícího procesu testování

3.1 Popis firmy

Stora Enso je předním poskytovatelem řešení v oblasti balení z obnovitelných materiálů. Dále se také zabývá biomateriály, dřevem a papírem. Jejími zákazníky jsou balírny, truhlářské firmy, vydavatelé knih, tiskárny, organizace pracující ve stavebním průmyslu apod.

Cílem firmy je nahradit neobnovitelné materiály novými produkty na bázi dřeva a jiných obnovitelných materiálů. Její produkty jsou tedy šetrné k životnímu prostředí a mají menší uhlíkovou stopu.

Stora Enso zaměstnává okolo 26 000 lidí ve více než 35 zemích světa a má dvě hlavní sídla. Jedno se nachází v Helsinkách, druhé ve Stockholmu. Tržby v roce 2015 byly okolo deseti miliard eur.

3.2 Stora Enso v Ostravě

SDCC Ostrava (Software Development Competence Center) bylo založeno v roce 2006 jako servisní centrum pro interní vývoj softwaru pro Stora Enso. V současné době jsou zde vyvíjeny a udržovány systémy pro podporu plánování, logistiku nebo prodej. Zákazníky jsou především pily a prodejní kanceláře Stora Enso. S počtem okolo 140 zaměstnanců je SDCC největší IT centrum Stora Enso.

3.3 CRM (Customer Relationship Management)

Chlebovský definuje Customer Relationship Management jako „interaktivní proces, jehož cílem je dosáhnout optimální rovnováhy mezi firemní investicí a uspokojením zákaznických potřeb. Optimum rovnováhy je determinováno maximálním ziskem obou stran.“ (Chlebovský, 2005, s. 23)

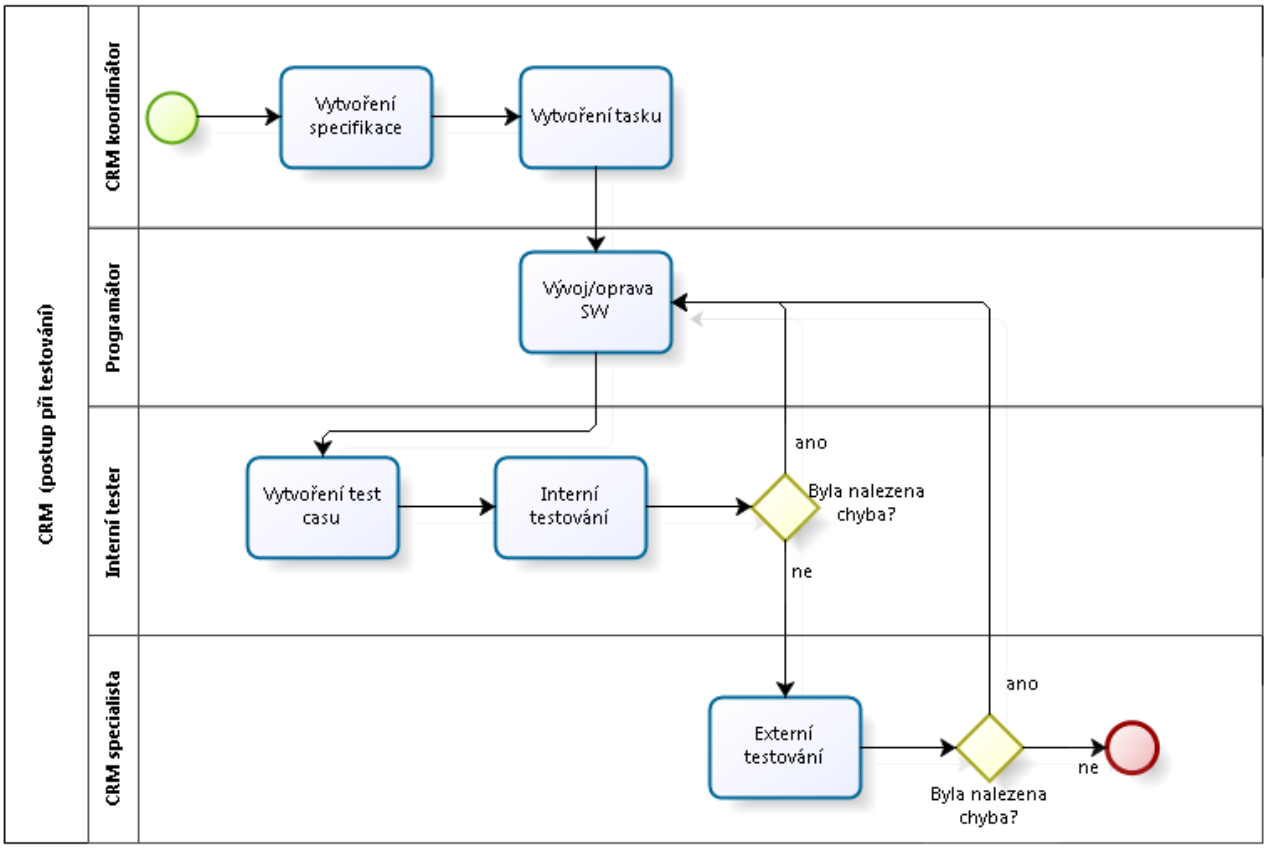
„Péče o zákazníky zahrnuje:

- trvalou aktualizaci zákaznických potřeb, motivací a zvyků,
- kvantifikaci přínosů základních funkcí CRM – marketingových, prodejních a servisních aktivit,
- využívání zákaznických znalostí a zkušeností při inovaci nabízených produktů,
- integraci marketingu, prodeje a zákaznické podpory v jednotný celek,

- využívání moderních nástrojů umožňujících podporu zákaznických potřeb a kvantifikaci přínosu CRM,
- trvalé udržování rovnováhy mezi marketingovými, prodejními a servisnímu aktivitami s cílem maximalizace zisku.“ (Chlebovský, 2005, s. 23)

Pro řízení vztahů se zákazníky toto oddělení používá Microsoft Dynamics CRM. Toto řešení umožňuje efektivně pracovat se zákazníky a vytvářet s nimi dobré vztahy. Microsoft Dynamics CRM obsahuje nástroje, se kterými mohou programátoři pracovat a vytvořit tak prostředí, které budou využívat jak zaměstnanci firmy, tak jejich zákazníci.

3.4 Současný postup při unit testování



Obr. 3.1 – postup při testování CRM

3.4.1 Vytvoření specifikace

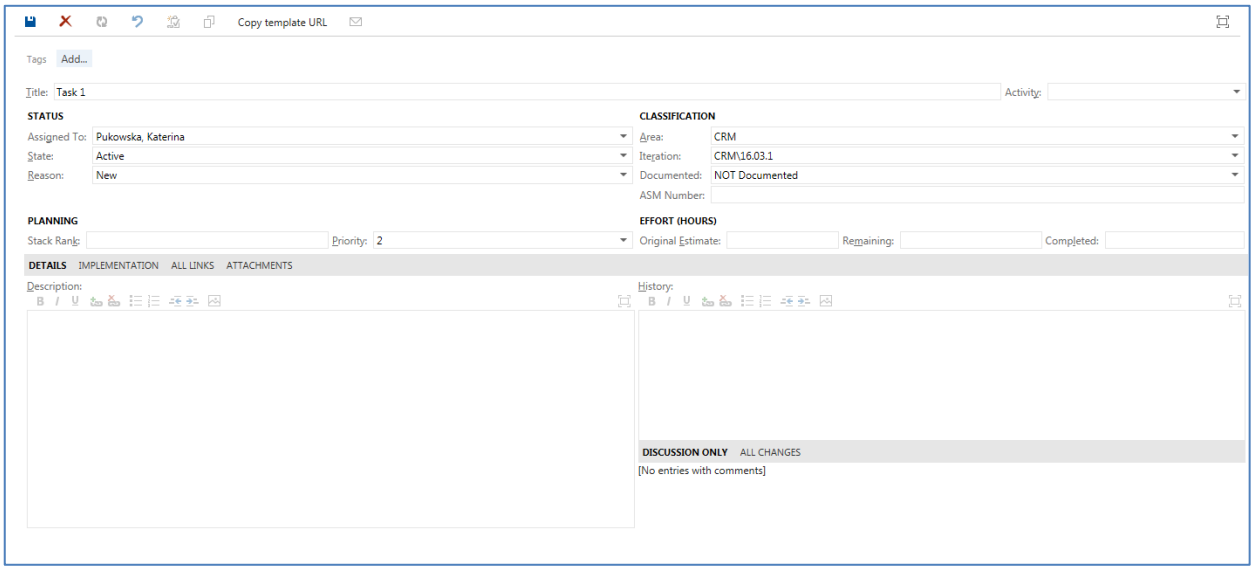
Požadavky na změnu v CRM pocházejí od jejich uživatelů. Do těch můžeme zařadit CRM specialisty, správce lesů, dopravce, marketingové specialisty apod. CRM experti poté tyto požadavky zašlou CRM koordinátorovi, který na jejich základě vytvoří specifikaci. Výsledkem je textový dokument, který zahrnuje tyto údaje:

- jméno člověka, které specifikaci vytvořil,
- název specifikace,
- název entity, které se změna týká,
- podrobný popis změny, která má být provedena,
- překlady, které musí být vykonány.

3.4.2 Vytvoření Tasku

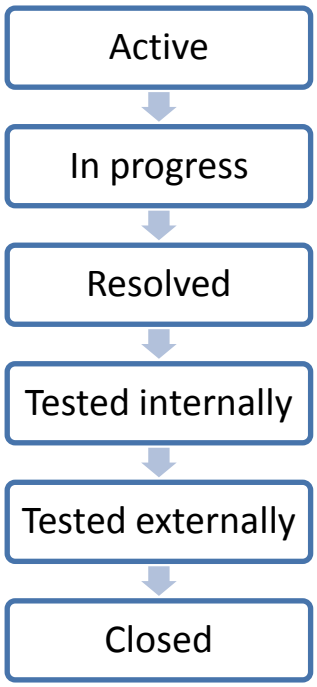
Na základě specifikaci se v Team Foundation Serveru (TFS) vytvoří Task neboli úkol, na kterém budou později pracovat vývojáři. Task obsahuje následující informace:

- název Tasku,
- jméno člověka (vývojáře), který momentálně na Tasku pracuje,
- aktuální stav práce,
- předpokládaná doba trvání,
- prioritu, severitu,
- detaily o Tasku,
- odkazy,
- diskuzi a pod.



Obr. 3.2 – vzhled Tasku v Microsoft Test Manageru

Jak již bylo zmíněno, při vytvoření Tasku se zadává jeho stav. Při práci v TFS jsou použity stavy Blocked, Active, In progress, Resolved, Tested internally, Tested externally, Closed. Již podle jejich názvu je patrné, že se jedná o jednotlivé etapy životního cyklu každého úkolu. Po vytvoření se Task nachází ve stavu Active a poté pokračuje dál, dokud se nedostane do stavu Closed. Pokud dojde během vývoje k situaci, kdy developer nemá dostatek informací a čeká na ně nebo nastanou jiné překážky, které neumožňují pokračovat v práci, přesune se Task do stavu Blocked. Jakmile je problém vyřešen, pokračuje se předešlým stavem. Ideální vývoj SW zobrazuje následující obrázek 3.3. Úkol zde nebyl blokován a ani testeři nenašli žádnou chybu.



Obr. 3.3 - ideální průběh vývoje SW

3.4.3 Vývoj

Po vytvoření specifikace a Tasku následuje samotný vývoj. Během této fáze je Task ve stavu In progress, po dokončení ve stavu Resolved. Při výběru úkolu, na kterém budou vývojáři pracovat, se dává přednost Tasku s vyšší prioritou. Nejvyšší priorita je 1. Tyto Tasky jsou kritické a mělo by se na nich ihned začít pracovat. Naopak nejméně důležité jsou úkoly s prioritou 4 a mohou se tedy odložit na pozdější termín.

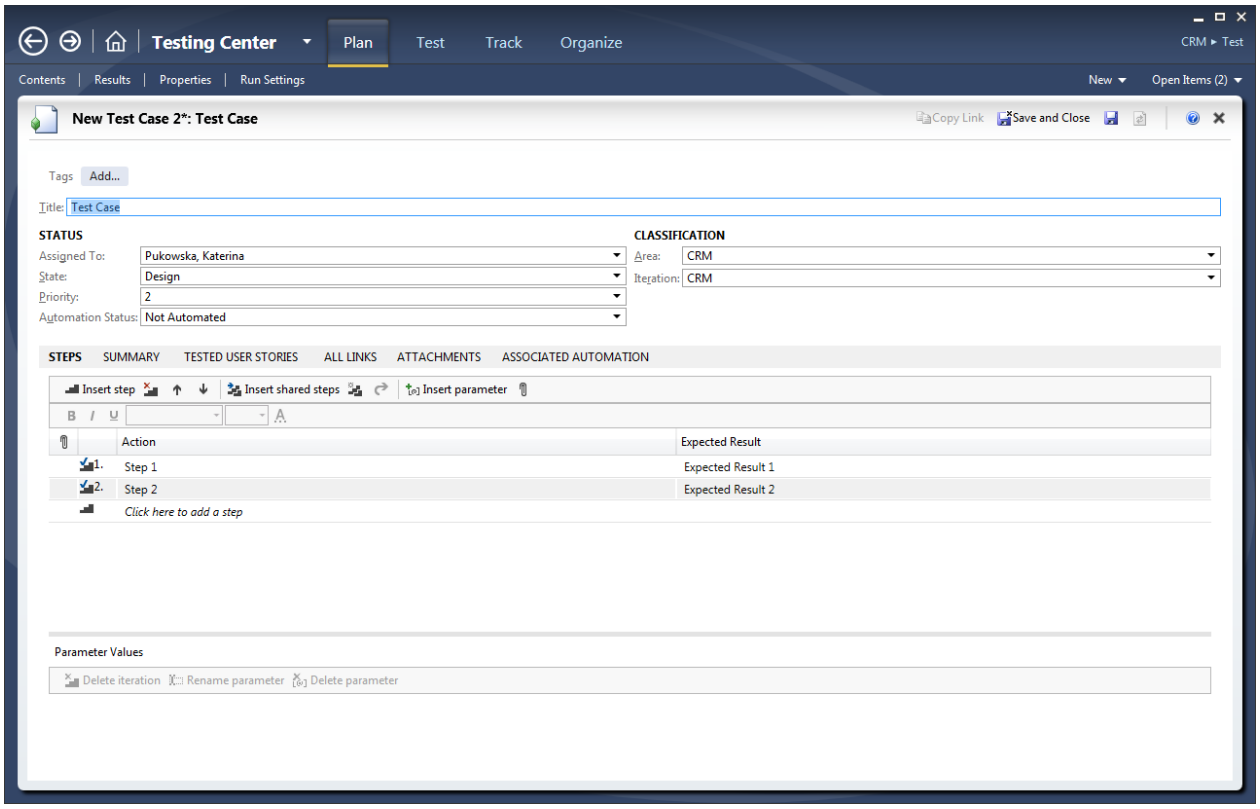
3.4.4 Vytvoření Test Casu

V době, kdy organizace s testováním teprve začínala, používal se k testování Excel. Byla předem vytvořena šablona, do které se zapisovaly všechny potřebné údaje. Na prvním listu byly obecné údaje, např. jméno autora, datum, jméno Tasku a jeho ID, iterace apod. V dalších listech pak byly popsané jednotlivé kroky, očekávané výsledky, snímky obrazovky, údaje, zda test prošel nebo ne a popřípadě krátké poznámky testera.

Později vznikl návrh na využití nástroje Microsoft Test Manager (MTM). Jelikož Microsoft nabízel třiceti denní zkušební verzi zdarma, mohlo se vyzkoušet, zda software bude vyhovovat naší organizaci. Bylo zjištěno, že MTM ulehčuje práci testerů, zvyšuje přehled v Test Casech a navíc spolupracuje s TFS, tudíž se rozhodlo, že se tento nástroj začne používat.

Vytvoření Test Casu má na starost tester, který je součástí CRM týmu v Ostravě. Dále ho budeme nazývat interní tester. Jeho postup při práci je následující: zvolí sekci Plan v Microsoft Test Manageru, poté vybere zařazení Test Casu a vytvoří nový testovací případ. Ten obsahuje:

- název test casu,
- přirazení testovacího případu konkrétnímu testerovi,
- stav,
- zařazení,
- jednotlivé kroky, které bude tester během testování vykonávat,
- očekávané výsledky apod.



Obr. 3.4 - MTM - vytvoření nového test casu

Název případu se skládá z čísla Tasku, který má být otestován, pomlčky a názvu Tasku. Tento způsob byl zvolen kvůli snadnějšímu vyhledávání v seznamu testovacích případů. Priorita a zařazení se volí stejné, jaké je v Tasku. u testovacího případu se také volí stav. Ten je však jiný, než u Tasku. Při vytváření nového případu se automaticky nastaví stav Design, další možnosti pak jsou Ready a Closed. Jelikož se zatím nevytváří žádné automatické testy je Automation Status nastaven vždy na stav Not Automated.

Jednotlivé kroky vycházejí ze specifikace, kterou najdeme v sekci Attachments v daném Tasku. Většinou se jedná o přílohu, která obsahuje Word nebo Excel. V některých případech je však úkol tak krátký, že se nevytváří žádný soubor se specifikací, ale pouze se přidá popisek v sekci Details.

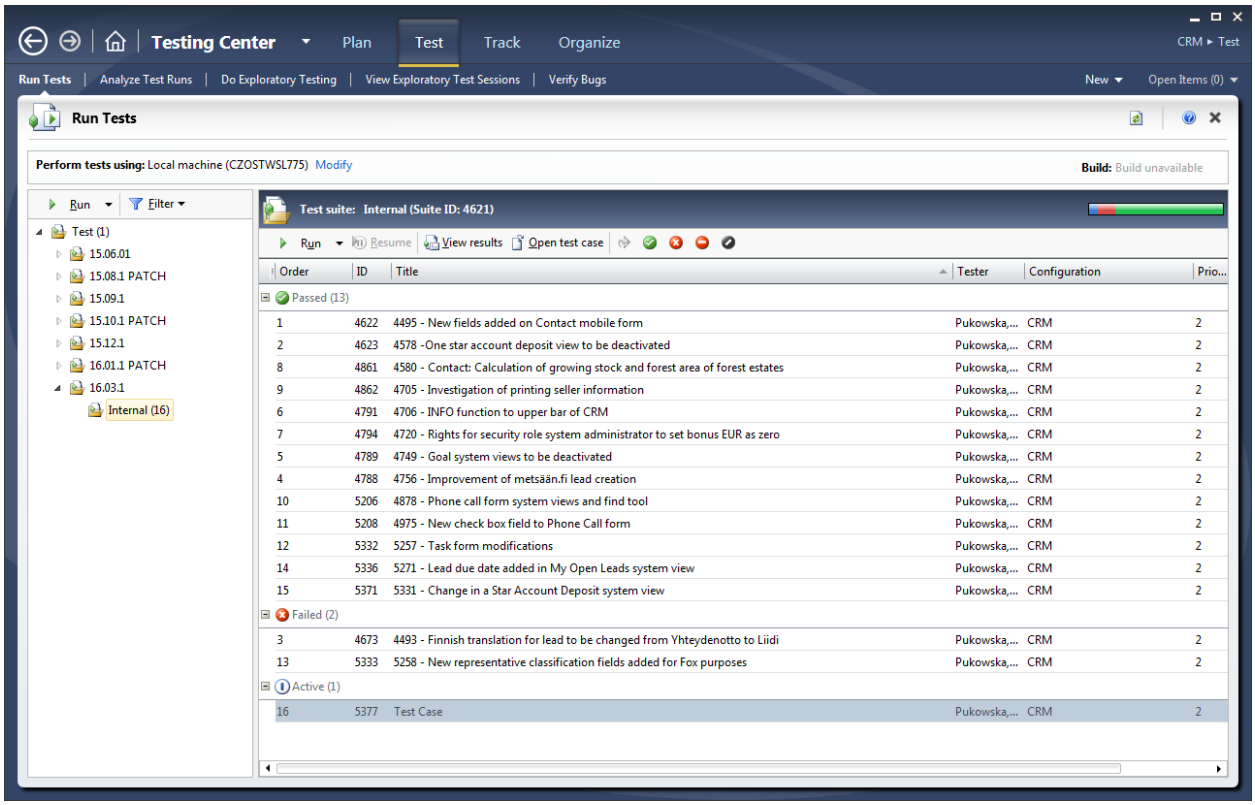
Vedle sloupce Action, kde jsou popsány jednotlivé kroky, můžeme nalézt sloupec Expected Result. Zde se píšou očekávané výsledky, ke kterým by měl tester dojít po provedení dané akce. Sloupec Expected Result se však nevyplňuje vždy, jelikož by to bylo velice zdlouhavé. Pokud je z dané akce patrné, co má tester očekávat, není třeba jej vyplňovat. Jako například můžeme uvést akci otevření prohlížeče. Je jasné, že očekávaným výsledkem je otevření prohlížeče. Jsou však případy, kde nemůžeme

odhadnout, co by systém měl po dané akci dělat nebo jak by měl vypadat. Například pokud akcí je kliknutí na nějaké tlačítko, nevíme, co by se následně mělo dít.

Jakmile je celý testovací případ hotový, provede se jeho prolinkovaní s Taskem, pro který byl vytvořen.

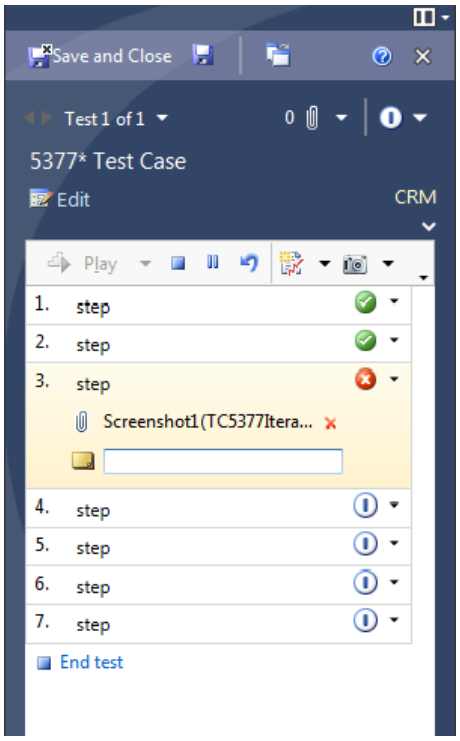
3.4.5 Interní testování

První fázi testování provádí interní tester, který si také sám připraví testovací případ. Poté musí přejít do sekce Test v Microsoft Test Manageru. Zde jsou zobrazeny všechny testy, které prošly, neprošly nebo jsou stále aktivní, neboť nebyly ještě spuštěny.



Obr. 3.5 – MTM - výpis všech vytvořených testovacích případů

Tester si vybere jeden z testovacích případů a spustí ho pomocí tlačítka Run. V levé části se zobrazí okno s jednotlivými kroky a jejich očekávanými výsledky. Na zbytku obrazovky pak může tester testovat, aniž by musel pořád přepínat okna. Následně tester označí jednotlivé kroky, zda byly úspěšné nebo ne. V případě, že dojde k selhání, je možnost napsat k danému kroku komentář nebo přidat obrázky zobrazující chybu.



Obr. 3.6 – vzhled MTM během testování

Jakmile jsou provedeny všechny kroky, test se ukončí. Uživateli se opět zobrazí původní okno se všemi záznamy. Test je označen zeleně jako Passed, pokud prošel nebo červeně jako Failed, pokud se v testu vyskytla alespoň jedna chyba. Tento testovací případ je možné si rozkliknout. Uživateli se zobrazí Test Result neboli výsledek testování. Zde jsou zobrazeny jednotlivé kroky, které prošly nebo ne, záznamy o předešlých testech a pod. Důležitou součástí Test Resultu je také možnost jeho prolinkování s příslušným Taskem. Pokud došlo k chybě a úkol je vracen programátorovi, stačí mu v Tasku rozkliknout sekci All links a zde najde příslušný Test Result, podle kterého pak může svou práci opravit. Následně pak Task vrátí zpět testerovi, který znova zkontroluje, jestli vše funguje jak má. Tento proces se stále opakuje, dokud není vše spraveno.

3.4.6 Externí testování

Pokud bylo vše během interního testování v pořádku, provede se ještě externí test CRM experty. Ti zkontrolují, zda je vše tak, jak si to představují, případně, zda se na něco nezapomnělo již při psaní specifikace. Pokud je vše v pořádku, je Task ukončen, tedy převeden do stavu Closed.

Externí testeři také testují celou aplikaci jako takovou, aby se zajistilo, že systém funguje správně. Tito testeři mají hlubší znalost o tomto systému, mohou tedy předvídat, co zákazník potřebuje a co by se mohlo zlepšit.

3.4.7 Vytvoření bugu

Pokud při testování byla nalezena chyba, ne vždy se v systému vytváří takzvaný bug. Často se totiž jedná o velice malý omyl, který může být opraven během krátké chvíle. Příkladem takové chyby může být překlep nebo chybějící překlad. Vytvoření bugu by trvalo déle než samotná korekce této chyby. V tomto případě se tedy Task vrátí programátorovi k opravě s důvodem, že test selhal. Pokud však programátoři usoudí, že potřebují delší dobu na tuto opravu, vytvoří se bug, na kterém následně pracují.

3.4.8 Závěr

V současnosti se projekt nachází ve stádiu, kdy testeři provádí pouze testy systémové, tudíž se dále budeme zabývat jen jimi. Jednotkové testy se obvykle nevytvářejí, v případě potřeby je provádí programátor sám. Akceptační testování se uskuteční až v další fázi, kdy většina systému bude hotová a mohou ho tak uživatelé vyzkoušet.

4 Návrh na inovaci procesu testování webových aplikací

Jelikož se s interním testováním CRM začalo nedávno, je zde spousta věci, které by se daly vylepšit. Tento proces neustálé probíhá a cílem tohoto snažení je nalézt chyby co nejefektivněji a zajistit tak lepší kvalitu výsledného produktu.

4.1 Vylepšení specifikace

Základem dobře odvedené práce při vývoji softwaru je především kvalitní specifikace produktu. Ta by měla podrobně popisovat všechny funkcionality, které se mají do systému přidat. Bohužel v praxi to tak ne vždy vypadá a programátoři musí zdlouhavě zjišťovat podrobnosti o připravovaných změnách.

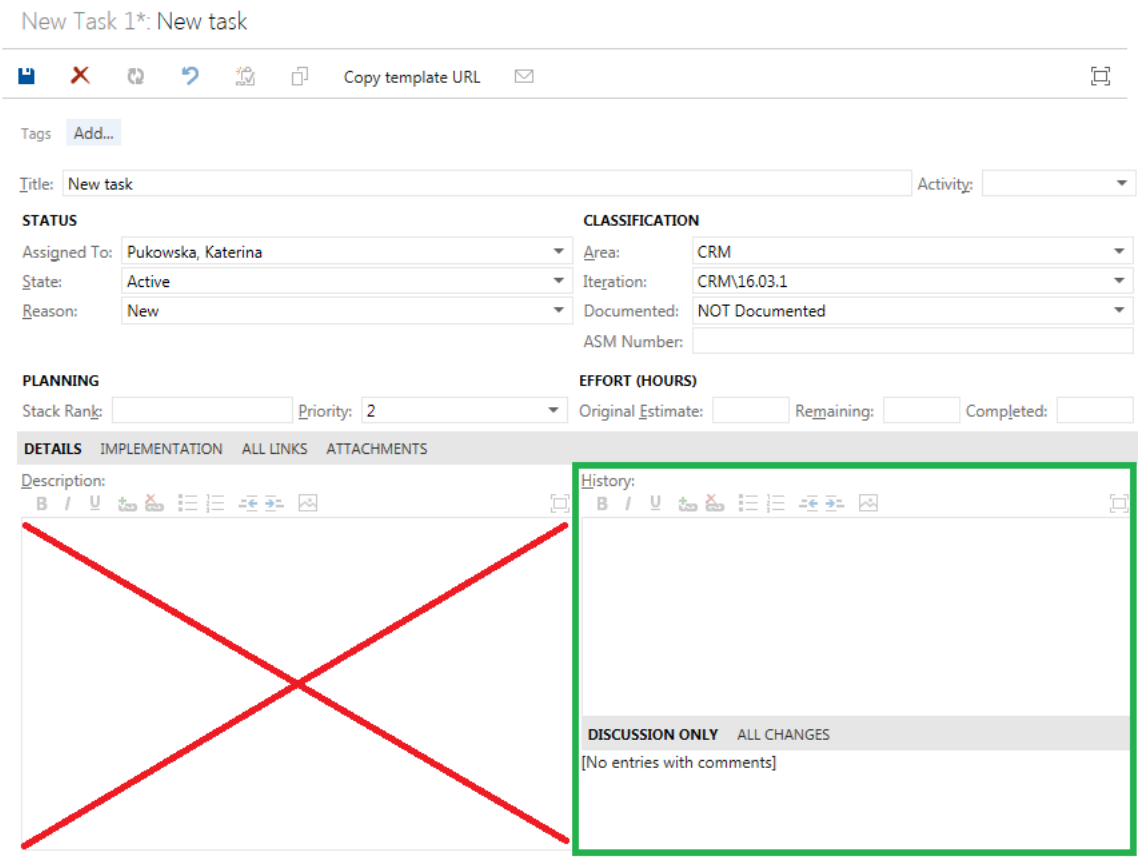
Pro vývoj CRM by bylo zajisté lepší, kdyby se specifikaci věnovalo více pozornosti a již od začátku bylo vše detailně definováno. Ulehčí to práci nejen programátorům, ale i testerům při psaní testovacích případů a především se tím ušetří spoustu času. Pokud se navíc jedná o popis složitých úkonů, kterým tester nerozumí, neboť na to nemá dostatečné znalosti, je dobré, pokud je současně se specifikací napsán i testovací případ.

4.2 Odstranění výsledku testování ze sekce Description

Dalším problémem je, že po otestování Tasku externím testerem dochází k situacím, kdy není vytvořen výsledek testování, pouze je vložen komentář do sekce Description v daném Tasku. To má za následek zmatení lidí, kteří Task následně čtou. Někteří testeři na začátek uvádějí své jméno, jiní pouze připíší text. Pokud však zde jméno není, nemusí být jasné, co je popis Tasku a co už poznámka přidaná testerem a může tak dojít k špatnému vyložení Tasku.

Řešením tohoto problému může být vytvoření textového dokumentu a popisem nalezené chyby, který se následně vloží jako Attachment. Bude tak zřejmé, kdo tento dokument vložil a kdy jej tam vložil. Programátoři si ho pak mohou jednoduše přečíst a zjistit, co je třeba napravit.

Druhou možností je vložení komentáře do sekce History. i u těchto komentářů je uvedeno jméno uživatele a čas vložení. Neměly by však být příliš dlouhé, aby se nesnížila přehlednost celého Tasku. Sekce History je znázorněna na následujícím obrázku 4.1.



Obr. 4.1 – sekce History v Tasku

4.3 Úprava specifikací

Podobný problém jako u vložení Test Resultu do sekce Description nastává, když je třeba změnit stávající specifikaci. Je třeba dát testerovi a ostatním členům týmu vědět, že se změnila a že tedy má testovat něco jiného. i zde se nabízí možnost přidání komentáře do sekce History. Tento krok však není úplně nejlepší, neboť při otevření Tasku tato historie není vidět a tester ji může přehlédnout. Historie se totiž zobrazuje ve spodní části Tasku a pokud ji chce uživatel vidět, musí posunout stránku dolu. Nejlepším řešením tedy je vytvoření kopie dokumentu, která bude mít stejný název, pouze zde bude připsáno, že se jedná o novou verzi. Nemůže tak dojít k situaci, kdy by si tester nevšiml změny a testoval by podle zastaralé specifikace.

4.4 Testování po nasazení na nový server

Stora Enso má pro vývoj vyhrazený svůj vlastní lokální testovací server. S ním mohou programátoři pracovat tak, jak potřebují. Provedené změny se zde poté otestují, a pokud je vše v pořádku, odešle se Task k externímu otestování. Tito testeři však testují na jiném serveru. Ten se bude používat k akceptačnímu testování. Může se tak stát, že

na lokálním serveru je vše v pořádku, ale k chybě došlo při implementaci změn na jiný server.

Z toho důvodu je dobré zavést kromě testování lokálního serveru i pravidelné testování všech ostatních serverů, kde se budou provádět změny. Jelikož se změny implementují jednou týdně, mělo by i toto testování probíhat jednou týdně a to hned po nasazení změn na server.

4.5 Zvýšení znalostí testerů

SDCC Ostrava v současnosti nemá žádné testovací oddělení, které by poskytovalo testery jiným oddělením podle potřeby. Pokud je někde tester potřeba, je najat pro konkrétní oddělení. Dochází tedy k tomu, že se používají různé testovací techniky a nástroje a i kvalita testování se může lišit. Někteří testeři totiž byli zaměstnání jako nováčci v tomto oboru.

Řešením může být sjednocení testování. Pokud by se na všech odděleních používal stejný nástroj na testování, jako je např. Microsoft Test Manager, mohli by být tito testeři snáz využiti i na jiných odděleních.

Dalším řešením může být školení zaměstnanců. V úvahu by se například mohla vzít jedna z nejznámějších certifikací na světě ISTQB (International Software Testing Qualifications Board). Zaměstnanci se tak díky ní mohou stát odborníky ve svém oboru. V roce 2015 mělo tuto certifikaci přes 410 000 lidí ve více než sto zemích světa. Certifikaci je možno získat ve třech základních úrovních:

- Foudation Level
- Advanced Level
- Expert Level.

Foundation Level je určen pro ty, kteří potřebují získat a prokázat základní znalosti pojmů spjaté s testováním softwaru. Můžeme sem tedy zařadit například testery, testovací analytiky, testovací konzultanty, test manažery apod. Dalšími vhodnými adepty jsou také lidé, kteří se testování přímo nevěnují, je však dobré, aby měli základní znalosti o testování. Mohou to být tedy projektoví manažeři, manažeři kvality, manažeři vývoje softwaru, ředitelé IT atd. u kandidátů na tuto certifikaci se předpokládá, že mají alespoň šesti měsíční zkušenost s testováním.

Po úspěšném složení zkoušky na úrovní Foundation , může uchazeč pokračovat na Advanced Level, který je určen pro ty, kteří se chtějí hlouběji porozumět testování. Pro získání této certifikace musí uchazeč mít Foudation cerfitikaci a splnit zkoušku. Teprve poté následuje Expert certifikace, která jde nejvíc do hloubky a je prakticky orientovaná. (ISTQB, 2016)

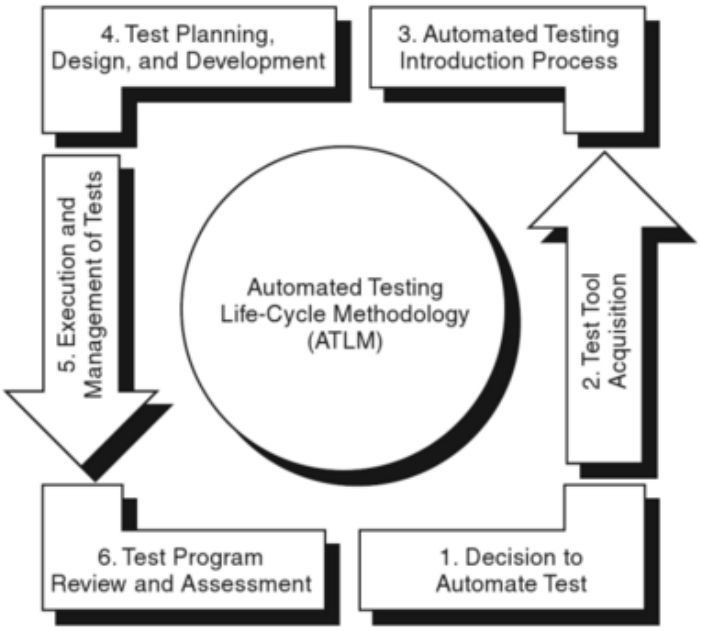
4.6 Zavedení automatického testování

Automatické testování může skvěle doplnit testování statické, je třeba však vždy pečlivě zvážit, pro co se vyplatí tyto testy psát. V současnosti se pro testování CRM nevytváří žádné automatické testy. Programátoři opravují vzniklé chyby, které jim nahlásí uživatelé nebo vylepšují systém. Tyto úkoly jsou však obvykle jednorázové a jejich automatizace by se nevyplatila. Provedení statického testu totiž zabere mnohem méně času než psaní testu automatického a pokud se tedy neprovádí víckrát, není výhodné jej psát.

Proces zavedení automatických testů se skládá z následujících částí:

1. rozhodnutí o zavedení automatických testů,
2. získání testovacích nástrojů,
3. příprava procesu testování,
4. plánování, návrh a vývoj testů,
5. provedení a správa testů,
6. kontrola a vyhodnocení testů. (Dustin)

Tento proces znázorňuje následující obrázek 4.2.



Obr. 4.2 – životní cyklus zavádění automatického testování. Zdroj: Dustin, 1999

4.6.1 Rozhodnutí o zavedení automatických testů

V této fázi by se měl celý tým sejít a ujasnit si očekávání, které mají na zavedení automatických testů a jaké benefity budou z toho plynout, pokud bude implementace úspěšná.

V CRM existují procesy, které jsou vždy stejné nebo se mění jen výjimečně. Jejich automatizací by se mohlo rychle zjistit, zda po provedení změn nebyla do systému zavlečena nějaká chyba a zda vše funguje jak má. Automatickým testem totiž můžeme detekovat nestandardní chování systému, na základě kterého pak tester je schopen snáz nalézt chybu. Pokud se tyto testy budou provádět pravidelně, jistě se testerovi vrátí čas, který vložil do psaní těchto testů. Díky rychlejšímu zjištění chyb tak může být zajištěna i lepší kvalita vyvíjeného softwaru.

Rozhodnutí o zavedení automatických testů záleží především na uvážení interního testera a také na schválení jeho nadřízeným pracovníkem. Pokud bude vybrán nástroj, který odpovídá požadavkům popsaným níže, je pak na něm, zda svůj čas využije k psaní těchto testů. Jelikož se v CRM se objevují procesy, které se opakují, bylo rozhodnuto automatické testy vytvořit.

4.6.2 Získání testovacích nástrojů

Pokud se rozhodneme, že využijeme některý z nástrojů na automatické testování, musí poté následovat jejich výběr. Je třeba si stanovit kritéria, které mají tyto nástroje splňovat a na základě toho vybrat ty nejlepší.

Požadavky na testovací nástroj

Pří výběru vhodného nástroje bychom měli dbát především na to, že CRM je primárně určeno pro prohlížeč Internet Explorer. Systém by zde měl fungovat co nejlépe, teprve následně se zjišťuje, zda nedochází k chybě v jiných prohlížečích. Díky tomuto nástroji tedy musíme být schopni pracovat s prohlížečem Internet Explorer, výhodou by však byla i práce s jinými prohlížeči.

Jelikož se k práci používá výhradně operační systém Windows, měl by ho tento nástroj podporovat. Není třeba hledat nástroje, které běží na jiných operačních systémech.

Další bezespornou výhodou aplikace by bylo, kdyby za ní firma nemusela platit a stačilo by využít softwaru, který už testeři vlastní nebo případně najít aplikace, které jsou zdarma. Jedním z nástrojů, který se již používá k testování, je Microsoft Test Manager. Ten umožňuje nastavit Automation Status v Test Casu. Díky tomu může tester ihned vidět, zda se jedná o test manuální nebo automatický. Jelikož chce firma zavést automatické testy, bylo by dobré, zvolit program, který tedy spolupracuje právě s MTM. Vše tak bude řádně zaznamenáno na jednom místě. Navíc pokud je napsán automatický test, který je následně připojen k Microsoft Test Manageru, může ho spustit kdokoli, aniž by musel mít znalosti v oblasti testování nebo programování.

Výhodou nástroje na testování bezesporu je, pokud s ním mohou pracovat jak začátečníci, tak pokročilí uživatelé. Pokud tester nemá dostatečné znalosti k psaní automatických testů, nabízí mnoho nástrojů na testování, které i přesto umožňují provádět automatické testy například pomocí nahrávání manuálního testování. Tuto nahrávku pak lze opakovatelně spustit a provést test, aniž by uživatel musel dělat stále ty stejné věci dokola. Problém je, že ne vždy je jednoduché vytvořit tuto nahrávku tak, aby fungovala bezchybně. Problém například může dělat vysouvací menu, které se ukáže jen po přesunutí kurzoru na určité místo. Pokud s kurzorem pohneme jinam, lišta se skryje a program tak nemusí najít prvky, se kterými je třeba pracovat. Po vytvoření

nahrávky je tedy nutné ji vždy zkontrolovat, případně opravit, což muže zabrat testerovi dost času.

Při výběru mají přednost nástroje, které jsou již uživateli ověřené. Pokud je s nimi spokojeno mnoho lidí, bude zřejmě vyhovovat i nám. Navíc v případě, že nástroj používá větší množství uživatelů, je snazší nalézt nápovědu na problém, se kterým si nevíme rady. Je pravděpodobné, že něco podobného už někdo řešil např. na nějakém fóru.

Selenium

Jako první možnost se nabízí nástroj Selenium. Jedná se o jeden z nejvíce používaných prostředků pro automatické testování webových aplikací. Tento open-source framework můžeme využít u všech prohlížečů, které podporují JavaScript, neboť právě tento nástroj je na něm založený. Selenium můžeme rozdělit do 4 částí:

- Selenium Grid,
- Selenium IDE,
- Selenium RC,
- Selenium Web Driver.

Selenium RC vytvořil Patrick Lightbody a Paul Hammant. Tento systém umožňuje ovládat webové prohlížeče lokálně nebo na jiných počítačích za použití téměř libovolného programovacího jazyka.

Selenium Grid je jedna z verzí Selenia, díky které můžeme provádět testy v několika prohlížečích a platformách zároveň, což nám může značně ušetřit čas.

Selenium IDE je add-on (doplněk) pro Firefox vyvinutý Shinya Kasatani. Tento nástroj umožňuje nahrávat práci testerů nebo vývojářů a následně si tento záznam znovu spustit.

Selenium Web Driver je framework pro všechny hlavní prohlížeče, díky kterému k nim můžeme přistupovat jako běžný koncový uživatel. Selenium Web Driver nahrazuje starší Selenium RC. Abychom mohli konkrétně využít Internet Explorer, je zapotřebí si stáhnout IE Driver, díky kterému pak můžeme prohlížeč otevřít. (Burns)

Právě Selenium Web Driver byl využit k automatickému testování systému CRM. Bohužel se ukázalo, že ne vždy funguje podle představ testera. Během testování často dochází k situaci, kdy je třeba něco vepsat do určitého pole. Toto psaní se však ukázalo jako zdlouhavé a tudíž ne příliš efektivní. Například takové napsání krátké zprávy „Ahoj, toto je můj první automatický test.“ do textového pole zabere velmi mnoho času.

Dalším problém nastal, pokud bylo potřeba kliknout na některý prvek, který neměl své vlastní ID. Poté bylo těžké ho lokalizovat, což také může souviset s nedostatečnou znalostí testera při využívání tohoto nástroje. Selenium umožňuje vyhledávat prvky na stránce pomocí parametru id, name, class, názvu tagu, textu linku nebo pomocí XPath a CSS selectoru. Pokud prvek na dané stránce nemá své vlastní id, je třeba si dávat pozor na to, podle čeho chceme vyhledávat. Pokud zvolíme např. název tagu, může se stát, že jich na stránce bude víc se stejným jménem.

Jako třetí velký problém se následně ukázalo celkové vyhledávání prvků na stránce. Pro tyto účely byl použit nástroj Internet Explorer Developer Tool (Nástroje pro vývojáře), které si můžeme rychle zobrazit zmáčknutím tlačítka F12. Práce s ním však není vždy příliš komfortní a může být zdlouhavá. CRM je totiž dosti složité a najít cestu k některým částem stránky není zcela jednoduché.

Výhodou Selenia je, že k psaní těchto testů můžeme využít Visual Studio a následně tyto testy propojit s Test Casem, který jsme si napsali v Microsoft Test Manageru. Tento nástroj však nedokáže s ním spolupracovat natolik, že by stačilo nahrát náš postup testování do MTM a automaticky ho následně převést na kód, který by se dal upravit.

Díky všem těmto chybám se rozhodlo od tohoto nástroje upustit a vybrat jiný, který bude více vyhovovat našim potřebám.

Coded UI

Jelikož testeři mají již licenci na Visual Studio Enterprise, dalším nástrojem, který vyhovoval námi zvoleným požadavkům je Coded UI. Podobně jako u Selenia můžeme díky tomuto frameworku provádět automatické testování. Coded UI bylo vyvinuto společností Microsoft. Velkou výhodou oproti Seleniu je fakt, že umožňuje jednodušší vyhledávání prvků na stránce. Uživatel si může spustit Coded UI Test Builder, který nám během chvíle dá mnoho informací o zvoleném elementu stránky. Tyto testy se také

ukázaly jako mnohem rychlejší a navíc zde lze vyhledávat snadno pomocí již zmíněného Coded UI Test Builderu, který nalezne mnohem více údajů o prvcích stránky, než například Internet Explorer Developer Tool. Uživatel může nalézt prvky pomocí:

- AutomationId,
- Name,
- LabeledBy,
- HelpText,
- AccessKey,
- AcceleratorKey,
- DisplayText,
- Source – v případě obrázků,
- Column Index – v případě datové tabulky.

4.6.3 Příprava procesu testování

Během přípravy procesu testování je důležité si především definovat, co budeme testovat. V našem případě je zapotřebí nalézt místa, která se příliš nemění, aby bylo výhodné pro ně automatické testy psát. Bohužel se stává, že dojde v systému k nějaké změně jako např. k přejmenování názvu některého z jeho tlačítek nebo k změně id. Pokud se takovéto tlačítko při provádění automatické testu používá, je třeba provést následnou opravu kódu tak, aby byl stále aktuální. Kdybychom tak neučinili, test nám neprojde. Pokud by docházelo k takovýmto změnám v CRM často, museli bychom svůj kód neustále přepisovat, což zabírá čas. Proto je lepší psát automatické testy pro ty části, které již jsou víceméně hotové.

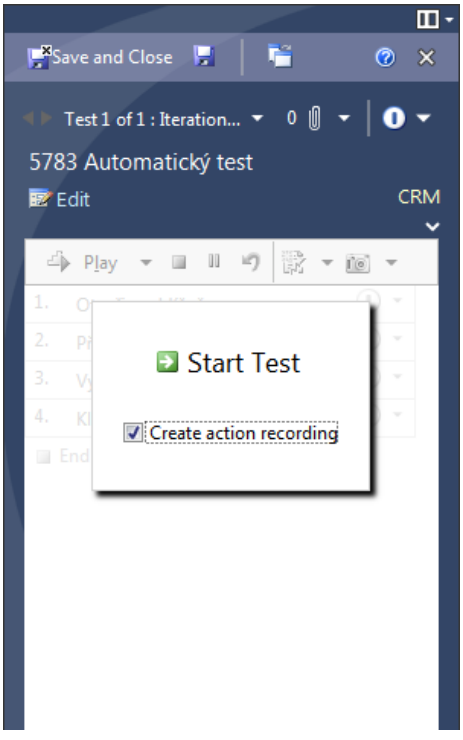
Kromě otestování jednotlivých entit CRM, zda obsahují všechny potřebné prvky, správné texty apod., se nabízí možnost otestovat JavaScripty a PlugIny. Právě u nich se totiž často stává, že po provedení některých změn v systému nepracují zdela správně nebo přestanou fungovat úplně.

4.6.4 Plánování, návrh a vývoj testů

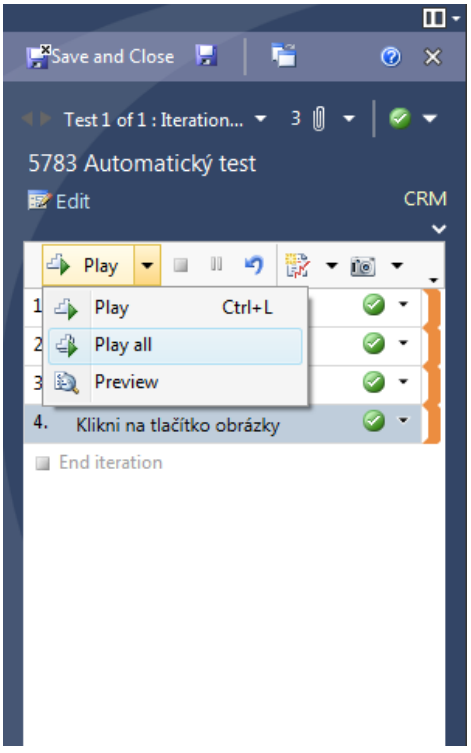
Jakmile máme vybrán a pořízen nástroj na testování a ujasněno, co chceme testovat, můžeme začít se samotným psaním testů, v našem případě s pomocí Coded UI. Jako první si vytvoříme klasický manuální test pomocí Microsoft Test Manageru, který bude sloužit jako návrh pro náš automatický test. Zde si detailně popíšeme jednotlivé

kroky, které je třeba provést a jejich očekávané výsledky. Nesmíme zapomenout zvolit Automation Status jako Planned a teprve poté test uložíme.

Pokud jsme úplní začátečníci, je zde možnost využít nahrávání naších kroků přes Microsoft Test Manager. Jednoduše test spustíme a před samotným startem zaškrtneme možnost „Create action recording“ a potvrdíme. Od té chvíle, vše co uděláme na našem počítači je nahráno. Musíme pamatovat na to, že je potřeba provést podstivě každý krok. Pokud máme již zapnutý prohlížeč a hned zde začneme testovat, nenahraje se nám jeho otevření. Po opětovném spuštění záznamu se tedy prohlížeč neotevře. Jakmile provedeme krok, který jsme si předem definovali v Test Casu, klikneme na potvrzovací tlačítko, čímž označíme, že akce byla úspěšně provedena. Tímto způsobem projdeme všechny kroky. Test ukončíme zmáčknutím „End iteration“. Poté bychom si měli celou nahrávku spustit znovu a zkontrolovat, jestli je vše v pořádku. Stačí vybrat v menu „Play all“ a vše se nám přehraje. Tento nástroj je sice jednoduchý a zvládne ho většina lidí se základními znalostmi práce s počítačem, ne vždy je však zcela přesný. Jsou situace, kdy se test špatně nahraje a je ho třeba i několikrát zopakovat.

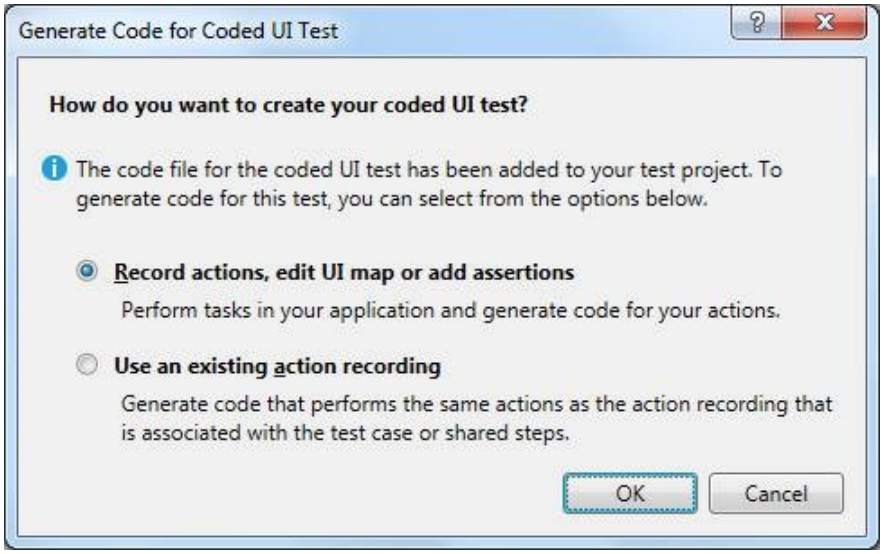


Obr. 4.3 – MTM – úvodní obrazovka



Obr. 4.4 – MTM – spuštění celé nahrávky

Jakmile je test nahraný, otevřeme si Visual Studio Enterprise. Vybereme možnost File – New - Project – Test – Coded UI Test. Po potvrzení vyskočí dialogové okno, které je zobrazeno v následujícím obrázku 4.5.

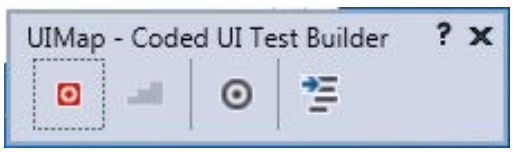


Obr. 4.5 – dialogové okno pro výběr typu Coded UI testu

Nyní máme na výběr 3 možnosti – vytvořit novou nahrávku, použit již existující nahrávku nebo okno zavřít a napsat si celý kód sami. My si popíšeme všechny tři možnosti.

Vytvoření nové nahrávky

Pro vytvoření nové nahrávky vybereme první možnost, tedy „Record actions, edit UI map or add assertions“ a zmáčkneme OK. Naše Visual studio se minimalizuje a objeví se Coded UI Test Builder. Jedná se o okno se čtyřmi základními tlačítky. První červené tlačítko slouží k nahrávání našeho testovaní, popřípadě k jeho stopnutí. Druhé tlačítko „Show Recorded Steps“ slouží k zobrazení kroků, které jsme si již nahráli. Pomocí třetího tlačítka můžeme přidat různá nastavení a zjistit si údaje o jednotlivých prvcích na stránce a nakonec poslední tlačítko slouží ke generování kódu.



Obr. 4.6 – Coded UI Test Builder

Test začnete stisknutím červeného tlačítka. To zmodrá a my můžeme začít pracovat. Poté se naše testování nahrává stejně jako u Microsoft Test Manageru. Test můžeme kdykoli pozastavit a zase v něm pokračovat. Jakmile jsme hotoví, necháme si vygenerovat kód. Objeví se nám okno, ve kterém můžeme ještě upravit název naší metody případně přidat popisek. Všechny nahrané akce jsou nyní uloženy v třídě nazvané UIMap. Nyní můžeme kód upravovat, musíme však myslet na to, že pokud provedeme změny v UIMap.Designer.cs a spustíme znovu Coded UI Test Builder, budou všechny naše provedené změny ztraceny, což je jedna z největších nevýhod používání nahrávání pro vytvoření automatických testů.

U takto nahraných testů je většinou zapotřebí přidat po vygenerování kódu další ověření. Například díky Coded UI můžeme zjistit, jestli se jednotlivé prvky na stránce rovnají, nerovnají, jestli jsou nebo nejsou null, jakým písmenem začínají nebo končí apod. Toto porovnání přidáme opět pomocí Coded UI Test Builderu. Zmáčkneme zde třetí tlačítko a poté na stránce nalezneme element, který budeme chtít otestovat. Zobrazí se nám okno s informacemi o tomto prvku. Pokud chceme například porovnávat, co obsahuje za text, vybereme vlastnost DisplayText a následně zvolíme možnost „Add Asertion“. Zde si můžeme vybrat jeden z komparátorů:

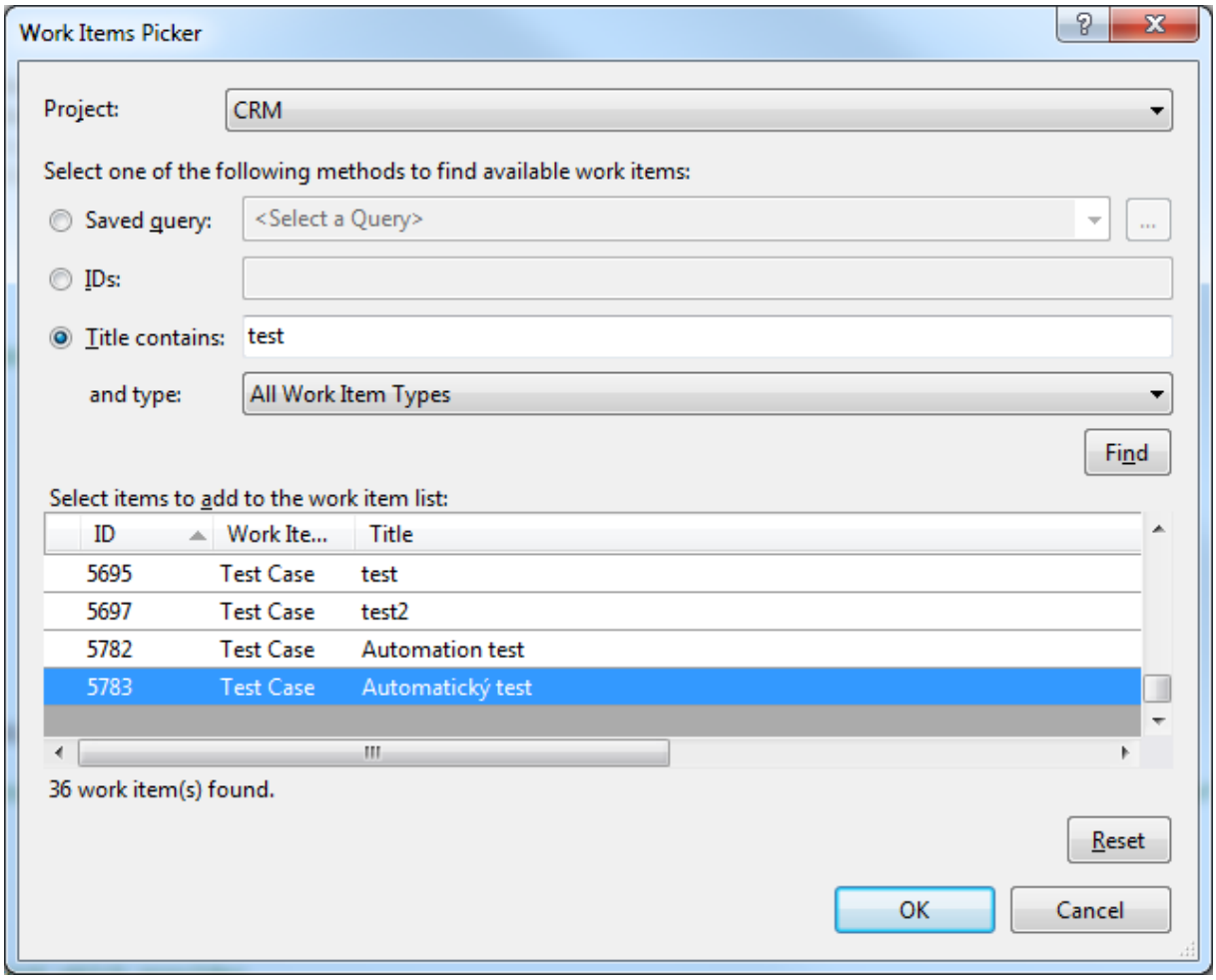
- AreEqual,
- AreNotEqual,
- Contains,
- Does Not Match,
- EndsWith,
- IsNotNull,
- IsNull,
- Matches,
- StartsWith.

Pokud dojde k situaci, že prvek neprojde kontrolou, můžeme zde ještě připsat chybovou hlášku. Jakmile jsme hotovi, opět si necháme vygenerovat kód.

Použití existující nahrávky

V případě, že chceme použít již existující nahrávku, kterou jsme vytvořili v Microsoft Test Manageru, musíme v dialogovém okně zvolit druhou možnost „Use an

existion action recording“. Následně je zapotřebí nalézt odpovídající Test Case, kde se nahrávka nachází, např. podle ID nebo části jeho názvu.



Obr. 4.7 – okno pro vyhledávání testovacího případu

Visual Studio vám vygeneruje kód, který je dobré před začátkem úprav spustit a zjistit tak, jestli vše funguje správně. Test spustíme pravým kliknutím myši dovnitř metody a výběrem možnosti „Run Test“. Po této kontrole může pokračovat úpravou kódu.

Opět klikneme pravým tlačítkem myši dovnitř metody a tentokrát vybereme Generate Code for Coded UI Test – Use Coded UI Test Builder. Otevře se nám Coded UI Test Builder, stejný jako u první možnosti. Díky němu můžeme přidat validace do již existujícího kódu, jak již bylo popsáno dříve.

Pokud chceme přidat další kroky do Visual Studia, zmáčkneme červené tlačítko na nahrávání a poté tlačítko „Generate Code“ a metodu pojmenujeme.

Vytvoření vlastního kódu

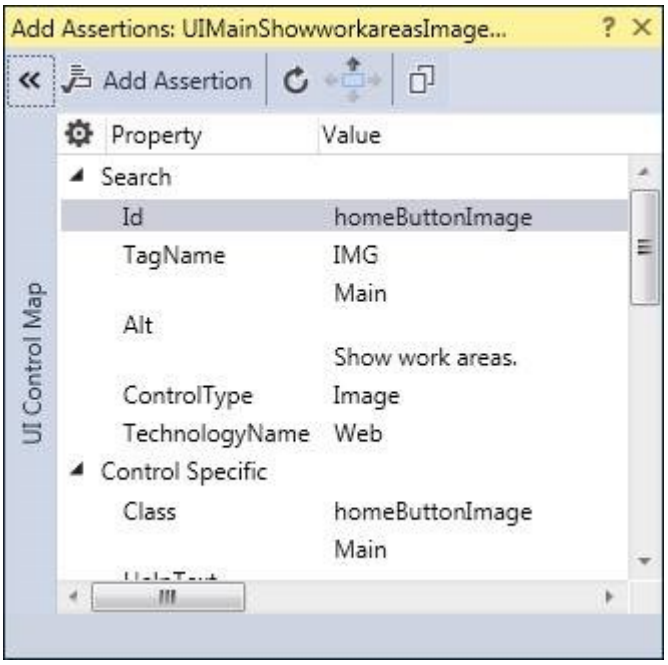
Nyní přejdeme ke třetí možnosti a tou je vytvoření svého vlastního kódu. V tomto případě se nevyužívá žádná nahrávka, ale tester si kód pro automatický test napíše zcela sám, proto je zapotřebí, aby měl alespoň základní znalosti o programování. Pokud používáme nahrávání, většinou je výsledný kód velmi dlouhý a rychle se v něm můžeme přestat orientovat. Výhodou tohoto psaní vlastního kódu je, že kód není tak obrovský a navíc si vše můžeme napsat podle sebe tak, jak nám to bude vyhovovat.

Jako první krok je samozřejmě otevření prohlížeče. Ten se klasicky otvírá v okně, které není přes celou naši obrazovku a je tedy zapotřebí jej maximalizovat. Dále je také třeba zjistit, na jakou webovou stránku se chceme dostat. Kód otevření prohlížeče pak může vypadat následovně:

```
[TestMethod]
0 references
public void CodedUITestMethod1()
{

    string url = "http://czostsrv39/FOXDEVCRM/main.aspx#897738523";
    BrowserWindow browser = BrowserWindow.Launch(url);
    browser.Maximized = true;
}
```

Další kroky už závisí čistě na tom, co chceme testovat. Obvykle na webových stránkách klikneme na nějaký odkaz, vepíše text do textového pole nebo klikneme a některé z tlačítek. Abychom to mohli provést, je třeba tyto prvky lokalizovat. K tomuto účelu použijeme opět Coded UI Test Builder. Klikneme na jeho třetí tlačítko a poté vybereme element stránky, o kterém potřebuje zjistit informace. Coded UI Test Builder nám vypíše všechny údaje, nejčastěji se však tyto prvky lokalizují podle ID.



Obr. 4.8 – okno s informacemi o nalezeném prvku stránky

Kromě ID je velmi důležité zjistit ControlType. Ten nám říká, jestli jde o link nebo jako například v předešlém obrázku 4.8 o obrázek (Image). Na základě toho se vytvoří instance třídy. u obrázku je to instance třídy HtmlImage, u tlačítka HtmlButton, u odkazu HtmlHyperlink apod.

Pro psaní automatických testů je dobré si vytvořit metody jako kliknutí na odkaz, kliknutí na tlačítko, kliknutí na obrázek apod. Tyto předem připravené funkce se mohou vložit do jedné třídy a následně je jen stačí zavolat, až je budeme potřebovat. Tyto metody jsou velmi podobné, mění se jen typ prvku, na který se funkce vztahuje a vlastnost, podle které chceme tento prvek vyhledat.

```
0 references
public void ClickLinkByID(UITestControl parent, string id)
{
    var link = new HtmlHyperlink(parent);
    link.SearchProperties.Add(HtmlHyperlink.PropertyNames.Id, id);
    Mouse.Click(link);
}

0 references
public void ClickImageName(UITestControl parent, string name)
{
    var link = new HtmlImage(parent);
    link.SearchProperties.Add(HtmlImage.PropertyNames.Name, name);
    Mouse.Click(link);
}
```

Další častou metodou pak je vložení textu do textového pole.

```
public void EnterText(UITestControl parent, string id, string value)
{
    var edit = new HtmlEdit(parent);
    edit.SearchProperties.Add(HtmlEdit.PropertyNames.Id, id);
    Mouse.Click(edit);
    edit.Text = value;
}
```

Pro budoucí automatické testování by mohlo být vytvořeno několik tříd, které by představovaly jednotlivé entity CRM. Dále by zde byla třída s metodami, jako jsou kliknutí na element, vložení textu apod. a hlavní třída, která by spojovala všechny entity a jejich testy. Stačilo by tedy spustit jeden test a provedly by se postupně všechny. Výsledek testů by pak bylo možno zapsat do konzole nebo jako textový dokument.

4.6.5 Provedení a správa testů, kontrola a vyhodnocení testů

Jakmile je vše připravené nalezneme ve Visual Studiu odpovídající testovací případ a k němu připojíme hlavní metodu obsahující náš test. Tímto se ihned změní Automation Status v Microsoft Test Manageru na Automated a z manuálního testu se tedy stane test automatický.

Následně můžeme přejít k samotnému provedení testu. Pokud jsme ho vytvořili nahráním, stačí ho spustit pomocí MTM. Vyhledáme si Test Case a poté zahájíme testování pomocí tlačítka Run. Test se automaticky provede a my poté můžeme vyhodnotit výsledky.

V případě, že nebyla použita nahrávka, ale tester si automatický test napsal sám může kromě spuštění v Microsoft Test Manageru využít Visual Studio. Spustí ho zde pravým kliknutím myši a výběrem možnosti Run Test. Výsledek se objeví v okně s názvem Test Explorer, které si může také rozrazit výběrem možnosti Test – Windows – Test Explorer. Během provádění testu by se nemělo hýbat s myší, ani nikam klikat, neboť toto naše chování může zapříčit neúspěch testování.

Pokud dojde k chybě, je třeba zjistit, co je její příčinou. Může se stát, že systém funguje správně, pouze jsme například nebyli připojeni k internetu nebo nebyla dostupná databáze. Může se také stát, že náš kód není aktuální a je třeba ho upravit tak, aby vyhovoval stávajícímu stavu CRM. Jak již bylo zmíněno dříve v sekci 4.6.3 Příprava procesu testování, mohlo dojít k změně názvu tlačítka apod. Pokud je však kód

správný a vše fungovalo, jak mělo a přesto test selhal, nelezli jsme chybu systému, kterou je třeba opravit.

To, jak často budeme testy provádět, záleží čistě na našem posouzení. Může to být denně, jednou týdně nebo například vždy po implementaci nové funkcionality.

5 Závěr

Cílem této práce bylo popsat teoretická východiska testování softwaru, provést analýzu stávajících procesů testování ve vybrané firmě a navrhnou jejich vylepšení.

Cíl této bakalářské práce byl splněn. V první části jsou shrnuty všechny důležité informace týkající se testování softwaru. Tento text může pomoci například začínajícím testerům zorientovat se v této oblasti. Mohou se zde dozvědět, jaké jsou jednotlivé techniky testování, kdo se o testy stará, co je to vůbec chyba softwaru apod.

V druhé části je popsána organizace a konkrétní oddělení, které bylo zkoumáno. Je zde také uvedena charakteristika celého procesu testování, který detailně znázorňuje obrázek 3.1. Tvrzení jsou pro lepší představivost doplněna o snímky z jednotlivých nástrojů především z Microsoft Test Manageru, který je hlavním testovacím prostředkem.

Třetí část obsahuje návrh na inovaci procesů. Tato oblast se skládá z kratších kapitol, jako je zlepšení specifikace a jejich úprava, odstranění testování ze sekce Description, testování po nasazení na nový server a návrh na zvýšení kvalifikace testerů. Kromě zvýšení kvalifikace zaměstnanců byla všechna tato zlepšení zavedena a v praxi dobře fungují. Na zvýšení kvalifikace zaměstnanců se pracuje a budoucnu by mělo následovat jejich školení. Poslední velká kapitola je věnována automatickému testování, které by se mělo stát běžnou součástí vývoje SW na oddělení CRM. V této kapitole je popsán celý životní cyklus zavedení automatického testování. Jako první nástroj byl k těmto účelům použit framework Selenium. Ten se ale příliš neosvědčil především z důvodu jeho pomalého psaní textu a také složitého vyhledávání elementů stánky. Proto byl nahrazen Coded UI. Ten je součástí Visual Studia Enterprise, který již zaměstnanci mají a navíc umožňuje vytvářet testy jak začátečníkům, tak pokročilejším uživatelům. Na automatickém testování se neustále pracuje. V současnosti je napsán jeden ukázkový test a nyní se zvažuje, co vše podrobí těmto testům.

6 Seznam použité literatury

ASH, Lydia. The *Web testing companion: the insider's guide to efficient and effective tests*. Indianapolis: Wiley, 2003. ISBN 04-714-3021-8.

BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů*. Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3.

BURNS, David. *Selenium 2 testing tools beginner's guide*. Birmingham: Packt, 2012. ISBN 978-1-84951-830-7.

DOMES, Martin. *Tvorba WWW stránek pro úplné začátečníky*. Brno: Computer Press, 2008. ISBN 978-80-251-2160-3.

DUSTIN, Elfriede, Jeff RASHKA and John PAUL. *Automated software testing: introduction, management, and performance*. New York: Addison-Wesley, 1999. ISBN 0-201-43287-0.

HAILPERN, Brent and Padmanabhan SANTHANAM. Software debugging, testing, and verification. *IBM Systems Journal.* 2002, vol. 41, iss. 1, p. 4-12. ISSN 0018-8670.

CHLEBOVSKÝ, Vít. *CRM: řízení vztahů se zákazníky*. Brno: Computer Press, 2005. ISBN 80-251-0798-1.

IEEE 1012: 2004. IEEE Standard for Software Verification and Validation. New York: Institute of Electrical and Electronics Engineers, 2005.

IEEE 610.12: 1990. IEEE Standard Glossary of Software Engineering Terminology. New York: Institute of Electrical and Electronics Engineers, 1990.

ISTQB. *Certifying Software Testers Worldwide.* [online]. © 2016 [cit. 2016-04-28]. Dostupné z: *http://www.istqb.org/*.

MCWHERTER, Jeff and Ben HALL. *Testing ASP.NET Web applications*. Indianapolis, IN: Wiley, 2010. ISBN 978-0-470-49664-0.

PATTON, Ron. *Software testing*. Indianapolis: Sams, 2001. ISBN 0-672-31983-7.

PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. ISBN 80-7226-636-5.

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.

7 Seznam zkratek

CRM	Customer Relationship Management (Řízení vztahů se zákazníky)
CSS	Cascading Style Sheets (Kaskádové styly)
HTML	Hypertext Markup Language (Značkovací jazyk pro hypertext)
IEEE	Institute of Electrical and Electronics Engineers (Institut pro elektrotechnické a elektronické inženýrství)
ISTQB	International Software Testing Qualifications Board
IT	Information technology (Informační technologie)
MTM	Microsoft Test Manager
SDCC	Software Development Competence Center
SQL	Structured Query Language (Standardizovaný strukturovaný dotazovací jazyk)
SW	Software
TFS	Team Foundation Server
UAT	User Acceptance Testing (Akceptační testování)
WWW	World Wide Web (Celosvětová síť)

Prohlášení o využití výsledků bakalářské práce

Prohlašuji, že

- jsem byla seznámena s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. – autorský zákon, zejména § 35 – užití díla v rámci občanských a náboženských obřadů, v rámci školních představení a užití díla školního a § 60 – školní dílo;

- beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, bakalářskou práci užít (§ 35 odst. 3);

- souhlasím s tím, že bakalářská práce bude v elektronické podobě archivována v Ústřední knihovně VŠB-TUO a jeden výtisk bude uložen u vedoucího bakalářské práce. Souhlasím s tím, že bibliografické údaje o bakalářské práci budou zveřejněny v informačním systému VŠB-TUO;

- bylo sjednáno, že s VŠB-TUO, v případě zájmu z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona;

- bylo sjednáno, že užít své dílo, bakalářskou práci, nebo poskytnout licenci k jejímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 3. 5. 2016

Pukowská

.....

Kateřina Pukowská